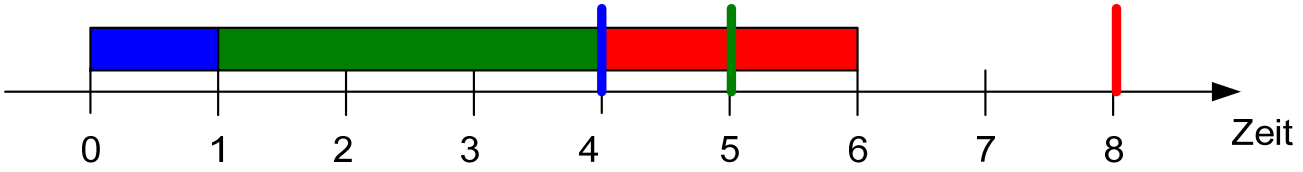
# Scheduling-Strategien (online, nicht-präemptiv) für Einprozessorsysteme

1. EDF: Einplanen nach Fristen (Earliest Deadline First): Der Prozess, dessen Frist als nächstes endet, erhält den Prozessor.
2. LST: Planen nach Spielraum (Least Slack Time): Der Prozess mit dem kleinsten Spielraum erhält den Prozessor.
   - Der Spielraum berechnet sich wie folgt:
     Deadline-(aktuelle Zeit + verbleibende Berechnungszeit)
   - Der Spielraum für den aktuell ausgeführten Prozess ist konstant.
   - Die Spielräume aller anderen Prozesse nehmen ab.

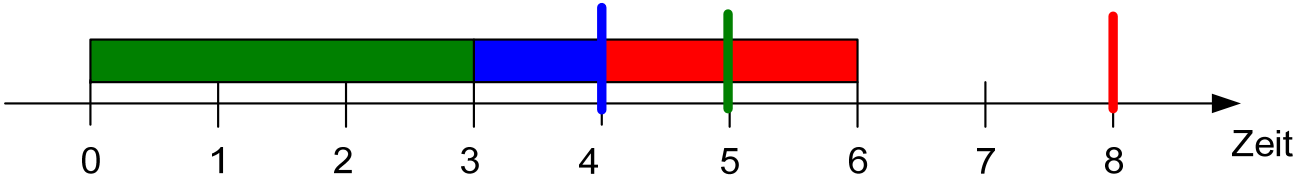
- Vorteil und Nachteile:
  - LST erkennt Fristverletzungen früher als EDF.
  - Für LST müssen die Ausführungszeiten der Prozesse bekannt sein.

# Beispiel

- 3 Prozesse:
  $P_1$: $r_1$=0; $e_1$=2; $d_1$=8;
  $P_2$: $r_2$=0; $e_2$=3; $d_2$=5;
  $P_3$: $r_3$=0; $e_3$=1; $d_3$=4;



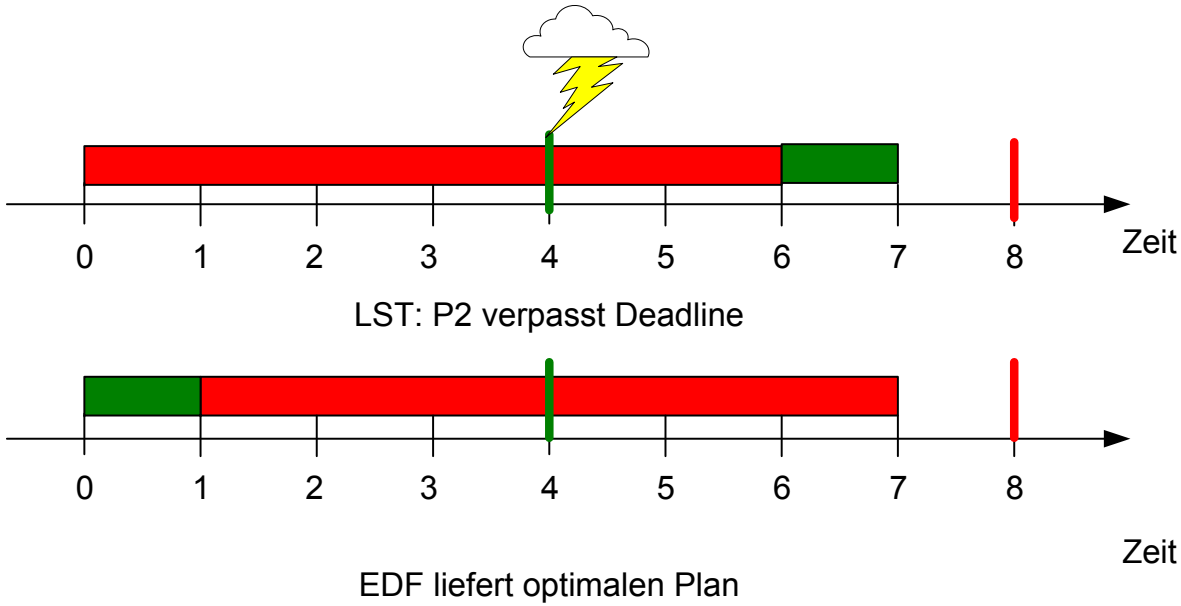

**Earliest Deadline First**




**Least Slack Time**

# Versagen von LST

- LST kann selbst bei gleichen Bereitzeiten im nicht-präemptiven Fall versagen.
- 2 Prozesse:
  $P_1$: $r_1$=0; $e_1$=6; $d_1$=8;
  $P_2$: $r_2$=0; $e_2$=1; $d_2$=4;




- Anmerkung: Aus diesem Grund wird LST nur in präemptiven Systemen eingesetzt. Bei Prozessen mit gleichen Spielräumen wird einem Prozess $\Delta$ eine Mindestausführungszeit garantiert.

# Optimalität von EDF

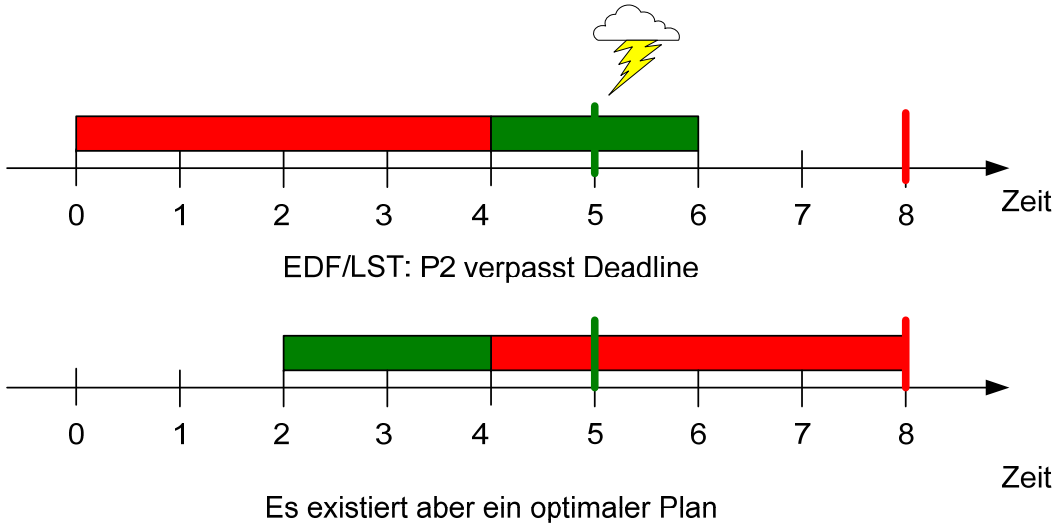
- Unter der Voraussetzung, dass alle Prozesse $P_i$ eine Bereitzeit $r_i=0$ besitzen und das ausführende System ein Einprozessorsystem ist, ist EDF optimal, d.h. ein zulässiger Plan wird gefunden, falls ein solcher existiert.
- Beweisidee für EDF: Tausch in existierendem Plan
  - Sie $Plan_x$ ein zulässiger Plan.
  - Sei $Plan_{EDF}$ der Plan, der durch die EDF-Strategie erstellt wurde.
  - Ohne Einschränkung der Allgemeinheit: die Prozessmenge sei nach Fristen sortiert, d.h. $d_i \leq d_j$ für i<j.
  - Idee: Schrittweise Überführung des Planes $Plan_x$ in $Plan_{EDF}$
    - $P(Plan_x,t)$ sei der Prozess, der von $Plan_x$ zum Zeitpunkt t ausgeführt wird.
    - $Plan_x(t)$ ist der bis zum Zeitpunkt t in $Plan_{EDF}$ überführte Plan ($\Rightarrow$ $Plan_x(0)=Plan_x$).

# Fortsetzung des Beweises

- Wir betrachten ein Zeitinterval $\Delta_t$.
- Zum Zeitpunkt t gilt:
  $i=P(Plan_{EDF},t)$
  $j=P(Plan_x,t)$
- Nur der Fall j>i ist interessant. Es gilt:
  - $d_i \leq d_j$
  - $t+\Delta_t \leq d_i$ (ansonsten wäre der $Plan_x$ nicht zulässig)
  - Da die Pläne bis zum Zeitpunkt t identisch sind und $P_i$ im $Plan_{EDF}$ zum Zeitpunkt t ausgeführt sind, kann der Prozess $P_i$ im $Plan_x$ noch nicht beendet sein.
    $\Rightarrow \exists\ t'>t+\Delta_t: (i=P(Plan_x,t')=P(Plan_x,t'+\Delta_t) \wedge t'+\Delta_t \leq d_i \leq d_j$
    $\Rightarrow$ Die Aktivitätsphase von $P_i$ im Zeitintervall $t'+\Delta_t$ und $P_j$ im Zeitintervall $t+\Delta_t$ können ohne Verletzung der Zeitbedingungen getauscht werden $\Rightarrow$ Übergang von $Plan_x(t)$ zu $Plan_x(t+\Delta_t)$

# Versagen von EDF bei unterschiedlichen Bereitzeiten

- Haben die Prozesse unterschiedliche Bereitzeiten, so kann EDF versagen.
- Beispiel: $P_1$: $r_1$=0; $e_1$=4; $d_1$=8 P2:$r_2$=2;$e_2$=2;$d_2$=5




- **Anmerkung**: Jedes prioritätsgesteuerte, **nicht präemptive** Verfahren versagt bei diesem Beispiel, da ein solches Verfahren nie eine Zuweisung des Prozessors an einen laufbereiten Prozess , falls ein solcher vorhanden ist, unterlässt.

# Modifikationen

- Die Optimalität der Verfahren kann durch folgende Änderungen sichergestellt werden:
  - Präemptive Strategie
  - Neuplanung beim Erreichen einer neuen Bereitzeit
  - Einplanung nur derjenigen Prozesse, deren Bereitzeit erreicht ist

  $\Rightarrow$ Entspricht einer Neuplanung, falls ein Prozess aktiv wird.
- Bei Least Slack Time müssen zusätzlich Zeitscheiben für Prozesse mit gleichem Spielraum eingeführt werden, um ein ständiges Hin- und Her Schalten zwischen Prozessen zu verhindern.
- Generell kann gezeigt werden, dass die Verwendung von EDF die Anzahl der Kontextwechsel in Bezug auf Online-Scheduling-Verfahren minimiert (siehe Paper von Buttazzo)

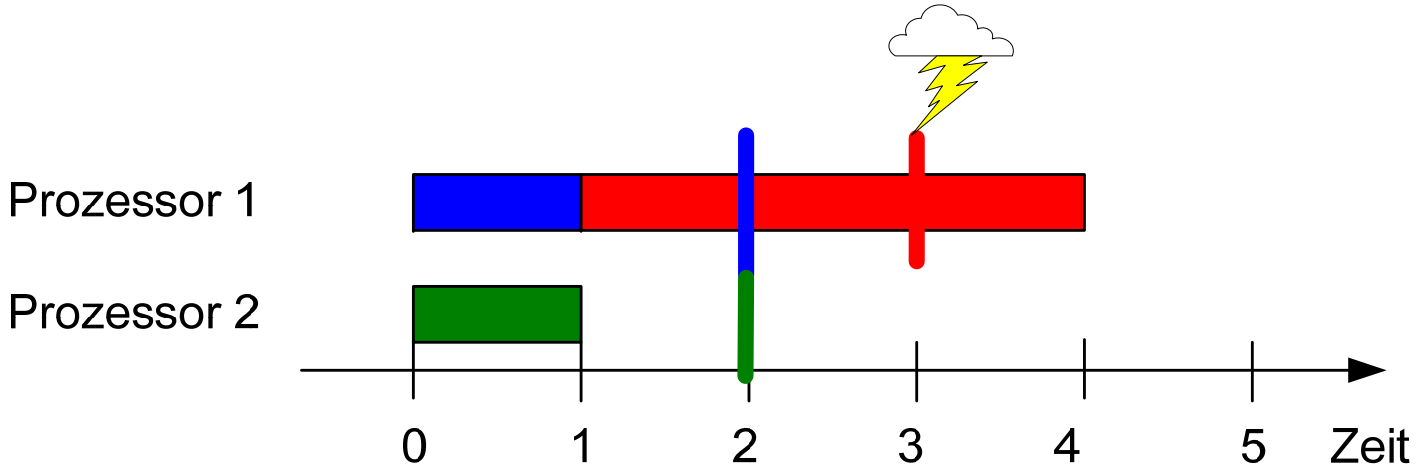
# Zeitplanung auf Mehrprozessorsystemen

# Zeitplanung auf Mehrprozessorsystemen

- Fakten zum Scheduling auf Mehrprozessorsystemen (Beispiele folgen):
  - EDF nicht optimal, egal ob präemptiv oder nicht präemptive Strategie
  - LST ist nur dann optimal, falls alle Bereitzeitpunkte $r_i$ gleich
  - korrekte Zuteilungsalgorithmen erfordern das Abarbeiten von Suchbäumen mit NP-Aufwand oder geeignete Heuristiken
  - Beweisidee zur Optimalität von LST bei gleichen Bereitzeitpunkten: Der Prozessor wird immer dem Prozess mit geringstem Spielraum zugewiesen , d.h. wenn bei LST eine Zeitüberschreitung auftritt, dann auch, falls die CPU einem Prozess mit größerem Spielraum zugewiesen worden wäre.

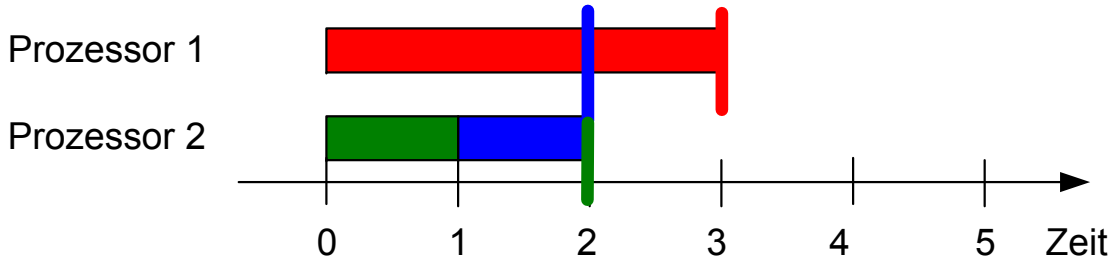
# Beispiel: Versagen von EDF

- 2 Prozessoren, 3 Prozesse:
  $P_1$: $r_1$=0; $e_1$=3; $d_1$=3;
  $P_2$: $r_2$=0; $e_2$=1; $d_2$=2;
  $P_3$: $r_3$=0; $e_3$=1; $d_3$=2;

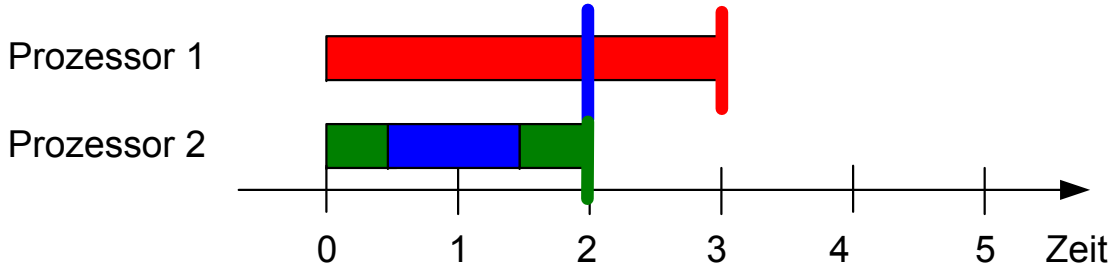



EDF-Verfahren: Deadline d1 wird verpasst

# Beispiel: Optimaler Plan und LST-Verfahren



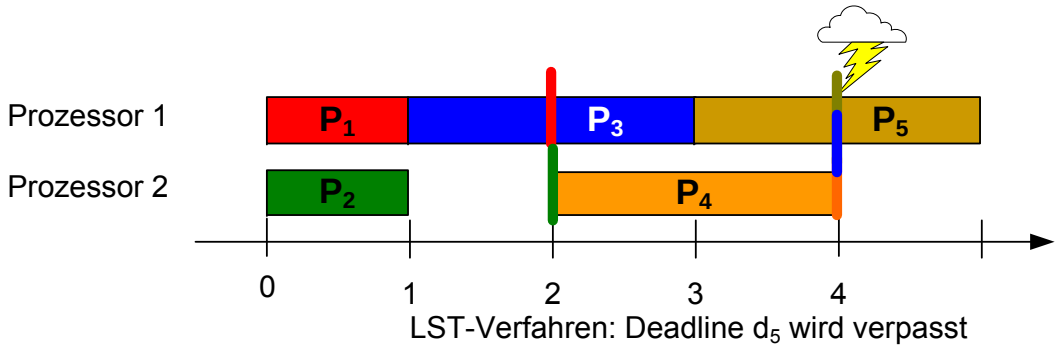

Optimaler Plan



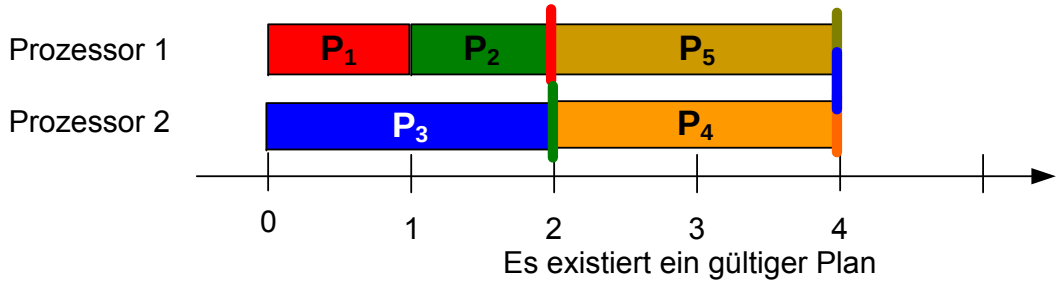

LST-Verfahren mit $\Delta_t = 0.5$

# Beispiel: Versagen von LST

- 2 Prozessoren, 5 Prozesse, $\Delta_t$=0,5:
  $P_1$: $r_1$=0; $e_1$=1; $d_1$=2;
  $P_2$: $r_2$=0; $e_2$=1; $d_2$=2;
  $P_3$: $r_3$=0; $e_3$=2; $d_3$=4;
  $P_4$: $r_4$=2; $e_4$=2;$d_4$=4;
  $P_5$: $r_5$=2; $e_5$=2;$d_5$=4;



LST-Verfahren: Deadline $d_5$ wird verpasst



Es existiert ein gültiger Plan

# Versagen von präemptiven Schedulingverfahren

- Jeder präemptiver Algorithmus versagt, wenn die Bereitstellzeiten unterschiedlich sind und nicht im Voraus bekannt sind.
  Beweis:
  - n CPUs und n-2 Prozesse ohne Spielraum (n-2 Prozesse müssen sofort auf n-2 Prozessoren ausgeführt werden) $\Rightarrow$ Reduzierung des Problems auf 2-Prozessor-Problem
  - Drei weitere Prozesse sind vorhanden und müssen eingeplant werden.
  - Die Reihenfolge der Abarbeitung ist von der Strategie abhängig, in jedem Fall kann aber folgender Fall konstruiert werden, so dass:
    - es zu einer Fristverletzung kommt,
    - aber ein gültiger Plan existiert.

# Fortsetzung Beweis

- Szenario:
  $P_1$: $r_1$=0; $e_1$=1; $d_1$=1;
  $P_2$: $r_2$=0; $e_2$=2; $d_2$=4;
  $P_3$: $r_3$=0; $e_3$=1; $d_3$=2;

  $\Rightarrow$ Prozess $P_1$ (kein Spielraum) muss sofort auf CPU1 ausgeführt werden.

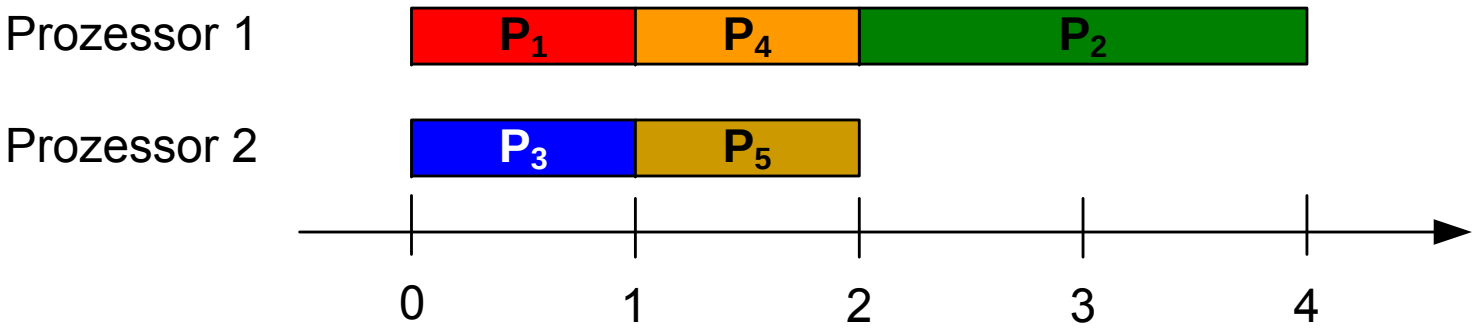
  $\Rightarrow$ Es gibt je nach Strategie zwei Fälle zu betrachten: P2 oder P3 wird zunächst auf CPU2 ausgeführt.

# 1. Fall

- $P_2$ wird zum Zeitpunkt 0 auf CPU2 ausgeführt.
  - Zum Zeitpunkt 1 muss dann $P_3$ (ohne Spielraum) ausgeführt werden.
  - Zum Zeitpunkt 1 treffen aber zwei weitere Prozesse $P_4$ und $P_5$ mit Frist 2 und Ausführungsdauer 1 ein.

  $\Rightarrow$ Es gibt drei Prozesse ohne Spielraum, aber nur zwei Prozessoren.

- Aber es gibt einen gültigen Ausführungsplan:

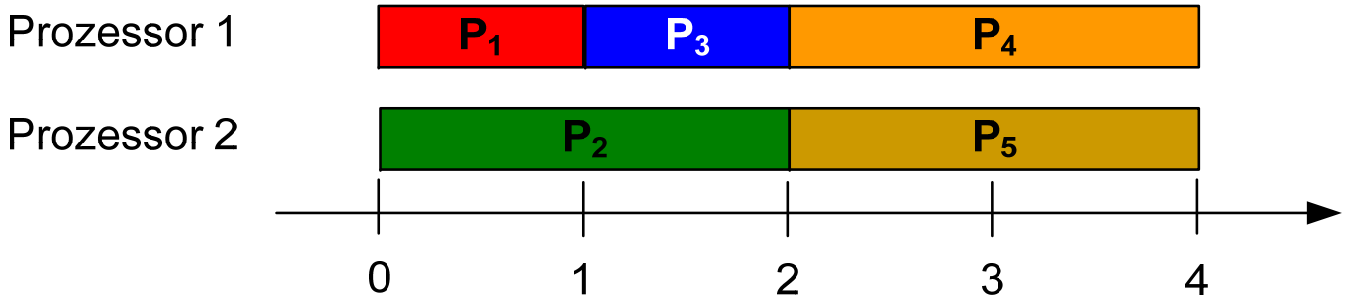


# 2. Fall

- $P_3$ wird zum Zeitpunkt 0 auf CPU2 ausgeführt.
  - Zum Zeitpunkt 1 sind $P_1$ und $P_3$ beendet.
  - Zum Zeitpunkt 1 beginnt $P_2$ seine Ausführung.
  - Zum Zeitpunkt 2 treffen aber zwei weitere Prozesse $P_4$ und $P_5$ mit Deadline 4 und Ausführungsdauer 2 ein.

  $\Rightarrow$ Anstelle der zum Zeitpunkt 2 noch notwendigen 5 Ausführungseinheiten sind nur 4 vorhanden.

- Aber es gibt einen gültigen Ausführungsplan:



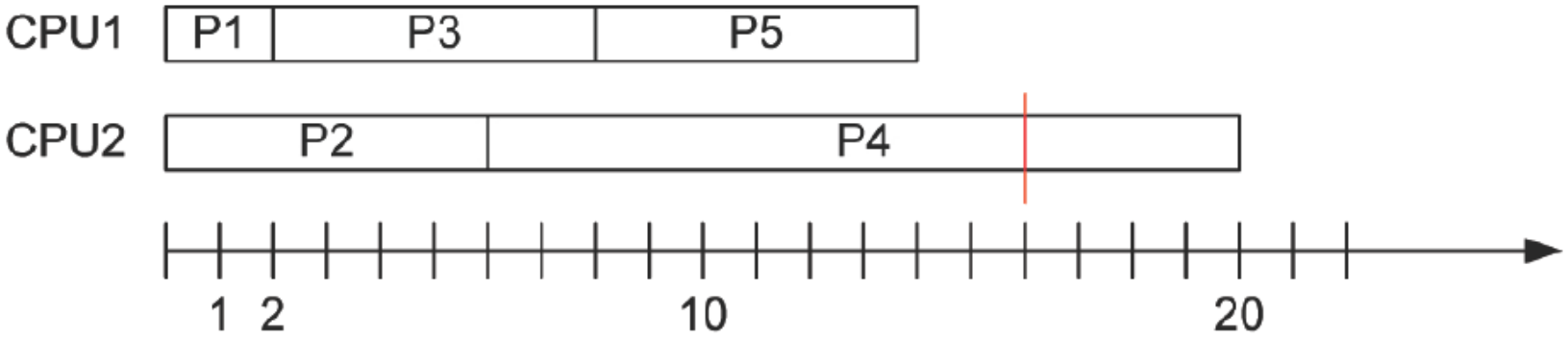
# Strategien in der Praxis

- Die Strategien EDF und LST werden in der Praxis selten angewandt. Gründe:
  - In der Realität sind keine abgeschlossenen Systeme vorhanden (Alarme, Unterbrechungen erfordern eine dynamische Planung)
  - Bereitzeiten sind nur bei zyklischen Prozessen oder Terminprozessen bekannt.
  - Die Abschätzung der Laufzeit sehr schwierig ist (siehe Exkurs).
  - Synchronisation, Kommunikation und gemeinsame Betriebsmittel verletzen die Forderung nach Unabhängigkeit der Prozesse.

# Ansatz in der Praxis

- Zumeist basiert das Scheduling auf der Zuweisung von statischen Prioritäten.
- Prioritäten werden zumeist durch natürliche Zahlen zwischen 0 und 255 ausgedrückt. Die höchste Priorität kann dabei sowohl 0 (z.B. in VxWorks) als auch 255 (z.B. in POSIX) sein.
- Die Priorität ergibt sich aus der Wichtigkeit des technischen Prozesses und der Abschätzung der Laufzeiten und Spielräume. Die Festlegung erfolgt dabei durch den Entwickler.
- Bei gleicher Priorität wird zumeist eine FIFO-Strategie (d.h. ein Prozess läuft solange, bis er entweder beendet ist oder aber ein Prozess höherer Priorität eintrifft) angewandt.
  **Alternative** Round Robin: Alle laufbereiten Prozesse mit der höchsten Priorität erhalten jeweils für eine im Voraus festgelegte Zeitdauer die CPU.

# Klausur SS 07 - Szenario




Startzeiten s: s(P1)=0; s(P2)=0; s(P3)=0; s(P4)=0; s(P5)=0;
Ausführungszeiten e: e(P1)=2; e(P2)=6; e(P3)=6; e(P4)=14; e(P5)=6;
Deadlines e: e(P1)=4; e(P2)=8; e(P3)=12; e(P4)=16; e(P5)=18;

a) Welches Schedulingverfahren wurde verwendet? Welche Änderungen würden sich ergeben, wenn das Verfahren präemptiv wäre?
b) Welche optimalen Schedulingverfahren existieren für Mehrprozessorsysteme?
c) Welche Voraussetzungen müssen für ein optimales Schedulingverfahren in Mehrprozessorsystemen erfüllt sein?
d) Zeichnen Sie einen unter Zuhilfenahme eines optimalen Schedulingplanes einen korrekten Ausführungsplan.
e) In der Praxis werden diese Schedulingverfahren nicht angewandt. Was spricht dagegen und welcher Ansatz wird stattdessen gewählt?

# Scheduling

Zeitplanen periodischer Prozesse

# Zeitplanung periodischer Prozesse

- Annahmen für präemptives Scheduling
  - Alle Prozesse treten periodisch mit einer Frequenz $f_i$ auf.
  - Die Frist eines Prozesses entspricht dem nächsten Startpunkt.
  - Sind die maximalen Ausführungszeiten $e_i$ bekannt, so kann leicht errechnet werden, ob ein ausführbarer Plan existiert.
  - Die für einen Prozesswechsel benötigten Zeiten sind vernachlässigbar.
  - Alle Prozesse sind unabhängig.
- Eine sehr gute Zusammenfassung zu dem Thema Zeitplanung periodischer Prozesse liefert Giorgio C. Buttazzo in seinem Paper „Rate Monotonic vs. EDF: Judgement Day“ (http://www.cas.mcmaster.ca/~downd/rtsj05-rmedf.pdf).

# Einplanbarkeit

- Eine notwendige Bedingung zur Einplanbarkeit ist die Last:
  - Last eines einzelnen Prozesses: $\rho_i = e_i * f_i$
  - Gesamte Auslastung bei n Prozessen:

$$\rho = \sum_{i=0}^{n} \rho_i$$

  - Bei m Prozessoren ist $\rho < m$ eine notwendige aber nicht ausreichende Bedingung.

# Zeitplanen nach Fristen

- **Ausgangspunkt**: Wir betrachten Systeme mit einem Prozessor und Fristen der Prozesse, die relativ zum Bereitzeitpunkt deren Perioden entsprechen, also $d_i=1/f_i$.
- **Aussage**: Die Einplanung nach Fristen ist optimal.
- **Beweisidee**: Vor dem Verletzen einer Frist ist die CPU nie unbeschäftigt $\Rightarrow$ die maximale Auslastung liegt bei 100%.
- Leider wird aufgrund von diversen Vorurteilen EDF selten benutzt.
- Betriebssysteme unterstützen selten ein EDF-Scheduling
  $\Rightarrow$ Die Implementierung eines EDF-Scheduler auf der Basis von einem prioritätsbasierten Scheduler ist nicht effizient zu implementieren (Ausnahme: zeitgesteuerte Systeme)

# Zeitplanung nach Raten

- Rate Monotonic bezeichnet ein Scheduling-Verfahren mit festen Prioritäten Prio(i), die sich proportional zu den Frequenzen verhalten.

  $\Rightarrow$ Prozesse mit hohen Raten werden bevorzugt. Das Verfahren ist optimal, falls eine Lösung mit statischen Prioritäten existiert. Verfahren mit dynamischen Prioritäten können allerdings eventuell bessere Ergebnisse liefern.

- Liu und Layland haben 1973 in einer Worst-Case-Analyse gezeigt, dass Ratenplanung sicher erfolgreich ist, falls bei n Prozessen auf einem Prozessor gilt:

$$\rho \leq \rho_{max} = n*(2^{1/n} - 1)$$

$$\lim_{n \to \infty} \rho_{max} = \ln 2 \approx 0{,}69$$

- Derzeit zumeist verwendetes Scheduling-Verfahren im Bereich von periodischen Prozessen.

# Scheduling

Planen abhängiger Prozesse

# Allgemeines zum Scheduling in Echtzeitsystemen

- Grundsätzlich kann der Prozessor neu vergeben werden, falls:
  - ein Prozess endet,
  - ein Prozess in den blockierten Zustand (z.B. wegen Anforderung eines blockierten Betriebsmittels) wechselt,
  - eine neuer Prozess gestartet wird,
  - ein Prozess vom blockierten Zustand in den Wartezustand wechselt (z.B. durch die Freigabe eines angeforderten Betriebmittels durch einen anderen Prozess)
  - oder nach dem Ablauf eines Zeitintervals, siehe z.B. Round Robin.
- Hochpriorisierte Prozesse dürfen in Echtzeitsystemen nicht durch unwichtigere Prozesse behindert werden $\Rightarrow$ Die Prioritätsreihenfolge muss bei allen Betriebsmitteln (CPU, Semaphore, Netzkommunikation, Puffer, Peripherie) eingehalten werden, d.h. Vordrängen in allen Warteschlangen.

# Präzedenzsysteme

- Zur Vereinfachung werden zunächst Systeme betrachtet, bei denen die Bereitzeiten der Prozesse auch abhängig von der Beendigung anderer Prozesse sein können.
- Mit Hilfe von Präzedenzsystemen können solche Folgen von voneinander abhängigen Prozessen beschrieben werden.
- Zur Beschreibung werden typischerweise Graphen verwendet:


- Der Nachfolgerprozess kann also frühestens beim Erreichen der eigenen Bereitzeit **und** der Beendigung der Ausführung des Vorgängerprozesses ausgeführt werden.

# Probleme bei Präzedenzsystemen

- Bei der Planung mit Präzedenzsystemen muss auch berücksichtigt werden, dass die Folgeprozesse noch rechtzeitig beendet werden können.
- Beispiel:
  $P_V$: $r_V=0$; $e_V=1$; $d_V=3$;
  $P_N$: $r_N=0$; $e_N=3$; $d_N=5$;
- Falls die Frist von $P_V$ voll ausgenutzt wird, kann der Prozess $P_N$ nicht mehr rechtzeitig beendet werden.

  $\Rightarrow$ Die Fristen müssen entsprechend den Prozessabhängigkeiten neu berechnet werden (Normalisierung von Präzedenzsystemen).

# Normalisierung von Präzedenzsystemen

- Anstelle des ursprünglichen Präzedenzsystems PS wird ein normalisiertes Präzedenzsystem PS' mit folgenden Eigenschaften:
  - $\forall$i: $e'_i = e_i$
  - $$\forall i : d'_i = \begin{cases} d_i, falls N_i = \emptyset \\ min(d_i, min(d'_q - e'_q | q \in N_i) \end{cases}$$

    wobei $N_i$ die Menge der Nachfolger im Präzedenzgraph bezeichnet und $d'_i$ rekursiv beginnend bei Prozessen ohne Nachfolger berechnet wird.
  - Falls die Bereitzeiten von externen Ereignissen abhängig sind, gilt $r'_i = r_i$. Sind die Bereitzeiten dagegen abhängig von der Beendigung der Prozesse, so ergeben sie sich aus dem konkreten Scheduling.

  eingeführt.

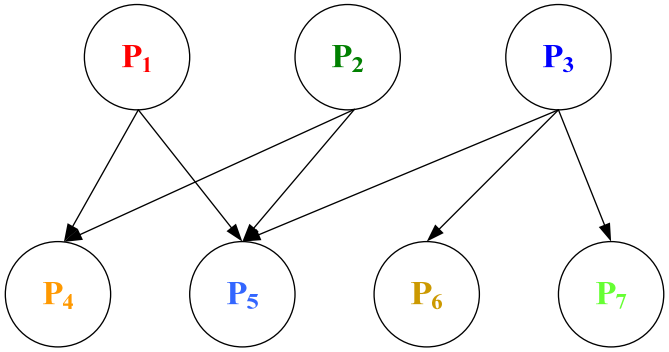
  $\Rightarrow$ Ein Präzedenzsystem ist nur dann planbar, falls das zugehörige normalisierte Präzedenzsystem planbar ist.

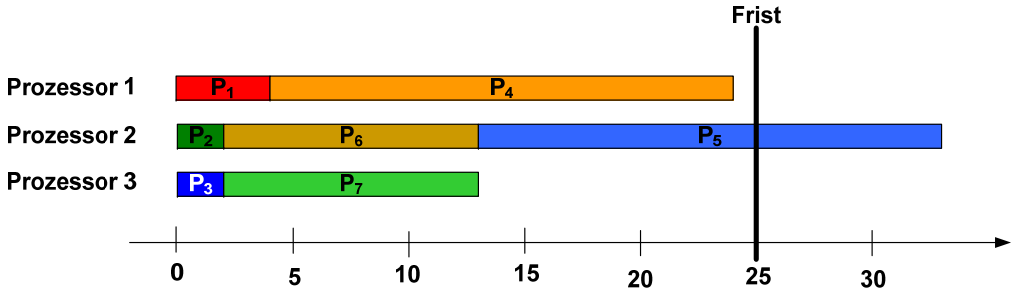
# Anomalien bei nicht präemptiven Scheduling

- Wird zum Scheduling von Präzedenzsystemen ein nicht präemptives prioritätenbasiertes Verfahren (z.B. EDF, LST) verwendet, so können Anomalien auftreten:
  - Durch Hinzufügen eines Prozessors kann sich die gesamte Ausführungszeit verlängern.
  - Durch freiwilliges Warten kann die gesamte Ausführungszeit verkürzt werden.

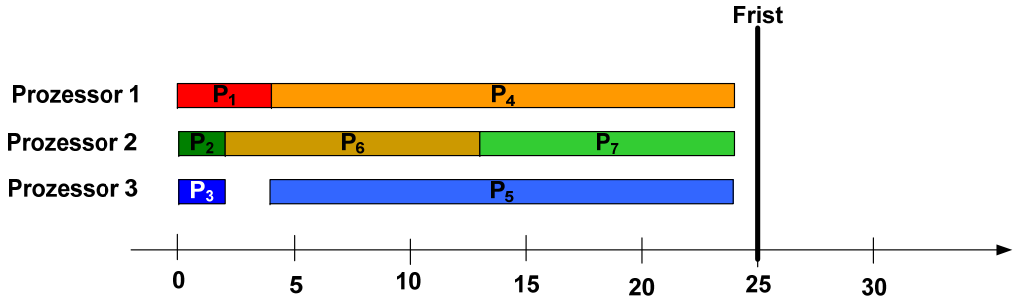
# Beispiel: Verkürzung durch freiwilliges Warten

- Beispiel: 3 Prozessoren, 7 Prozesse ($r_i$=0, $e_1$=4; $e_2$=2; $e_3$=2; $e_4$=20; $e_5$=20; $e_6$=11; $e_7$=11, $d_i$=25), Präzedenzgraph:




Prioritätenbasiertes Scheduling

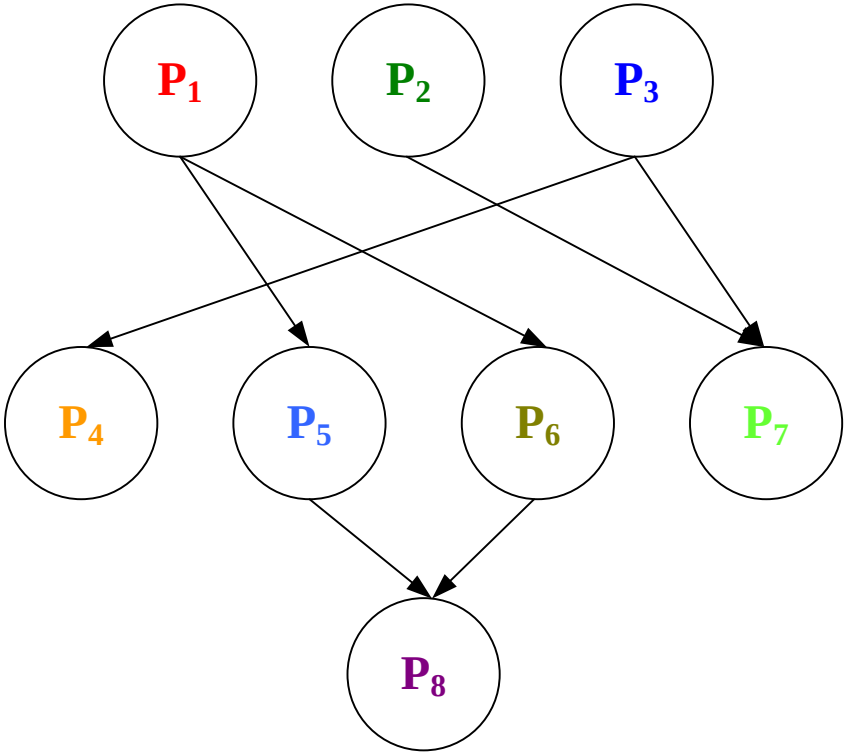



Optimaler Plan

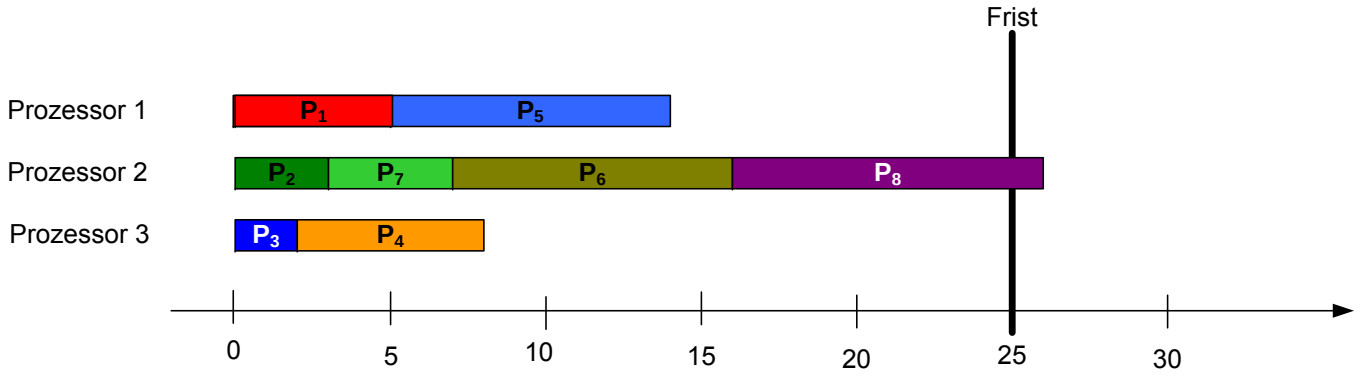
# Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II

- Beispiel:
  - 2 bzw. 3 Prozessoren
  - 8 Prozesse:
    - Startzeiten $r_i=0$
    - Ausführungszeiten
      $e_1=5$;
      $e_2=3$;
      $e_3=2$;
      $e_4=6$;
      $e_5=9$;
      $e_6=9$;
      $e_7=4$,
      $e_8=10$
    - Frist: $d_i=25$

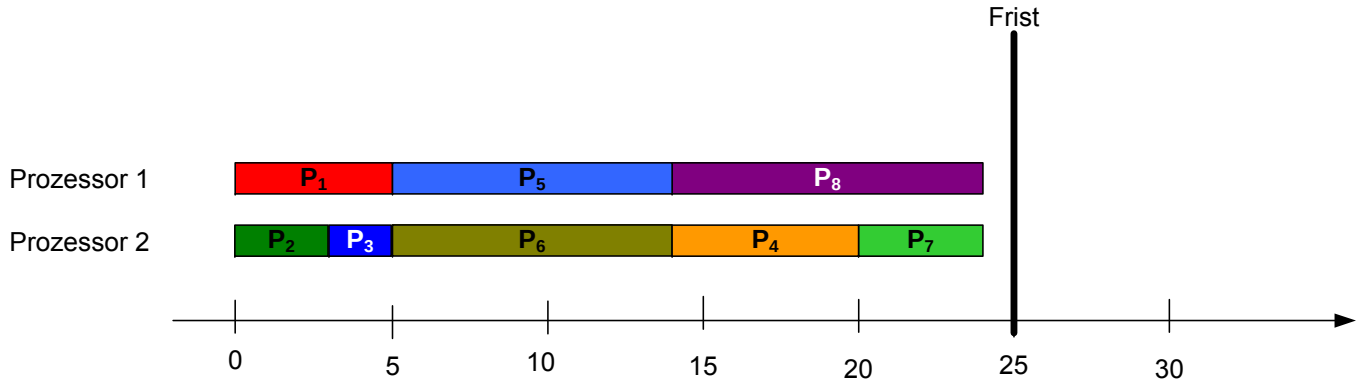
- Präzedenzgraph:



# Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II




Prioritätenbasiertes Scheduling (LST) auf 3 Prozessoren




Prioritätenbasiertes Scheduling (LST) auf 2 Prozessoren

# Scheduling

Problem: Prioritätsinversion

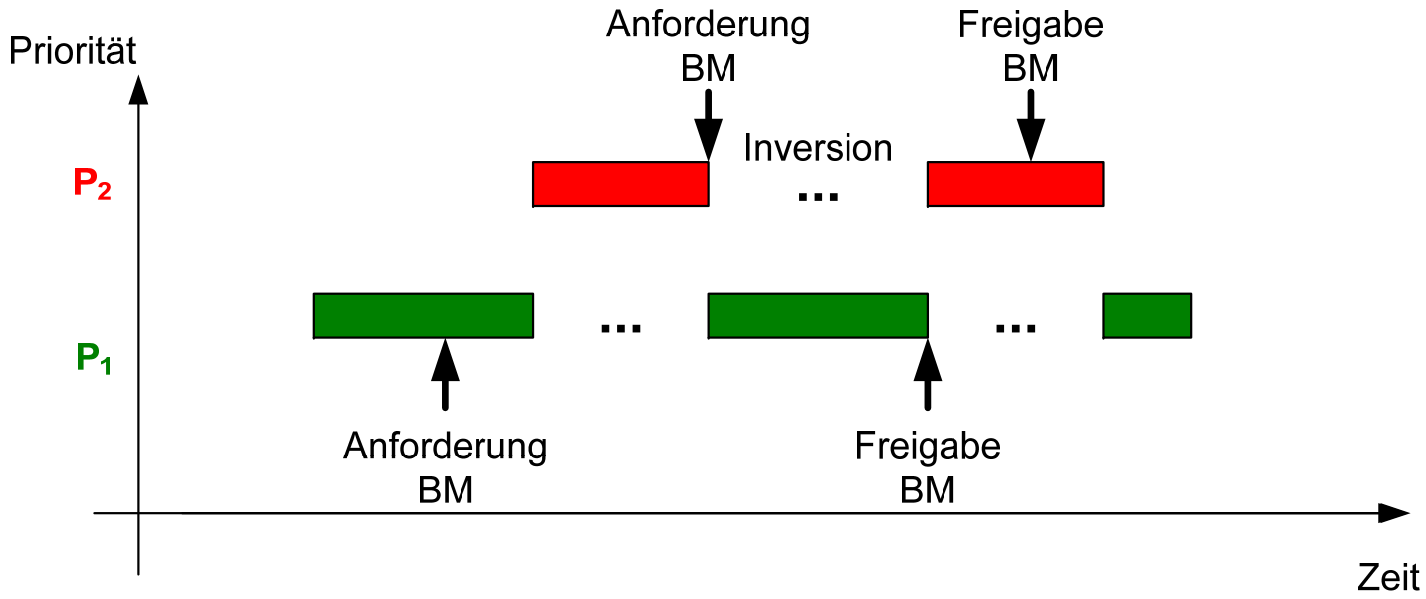
# Motivation des Problems

- Selbst auf einem Einprozessoren-System mit präemptiven Scheduling gibt es Probleme bei voneinander abhängigen Prozessen.
- Abhängigkeiten können diverse Gründe haben:
  - Prozesse benötigen Ergebnisse eines anderen Prozesses
  - Betriebsmittel werden geteilt
  - Es existieren kritische Bereiche, die durch Semaphoren oder Monitoren geschützt sind.
- Gerade aus den letzten zwei Punkten entstehen einige Probleme:
  - Die Prozesse werden unter Umständen unabhängig voneinander implementiert $\Rightarrow$ das Verhalten des anderen Prozesses ist nicht bekannt.
  - Bisher haben wir noch keinen Mechanismus zum Umgang mit blockierten Betriebsmitteln kennengelernt, falls hochpriore Prozesse diese Betriebsmittel anfordern.

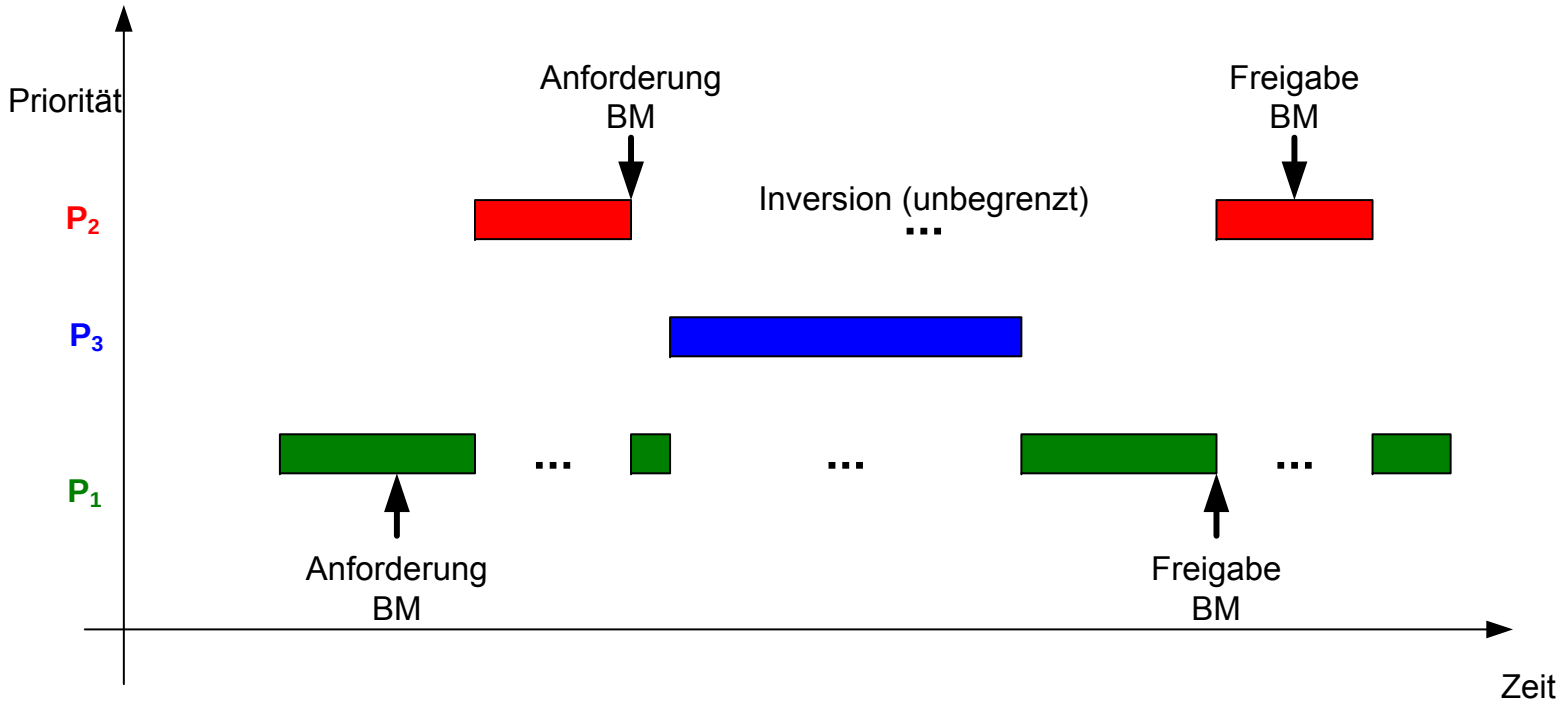
# Prioritätsinversion

- **Definition**: Das Problem der **Prioritätsinversion** bezeichnet Situationen, in denen ein Prozess mit niedriger Priorität einen höherpriorisierten Prozess blockiert.
- Dabei unterscheidet man zwei Arten der Prioritätsinversion:
  - **begrenzte** (bounded) Prioritätsinversion: die Inversion ist durch die Dauer des kritischen Bereichs beschränkt.
  - **unbegrenzte** (unbounded) Prioritätsinversion: durch weitere Prozesse kann der hochpriorisierte Prozess auf unbestimmte Dauer blockiert werden.
- Während das Problem der begrenzten Prioritätsinversion aufgrund der begrenzten Zeitdauer akzeptiert werden kann (muss), ist die unbegrenzte Prioritätsinversion in Echtzeitsystemen unbedingt zu vermeiden.

# Begrenzte Inversion



# Unbegrenzte Inversion



# Reales Beispiel: Mars Pathfinder

- **System**: Der Mars Pathfinder hatte zur Speicherung der Daten einen Informationsbus (vergleichbar mit Shared Memory). Der Informationsbus war durch einen binären Semaphore geschützt. Ein Bus Management Prozess verwaltete den Bus mit hoher Priorität.
  Ein weiterer Prozess war für die Sammlung von geologischen Daten eingeplant. Dieser Task lief mit einer niedrigen Priorität. Zusätzlich gab es noch einen Kommunikationsprozess mittlerer Priorität.
- **Symptome**: Das System führte in unregelmäßigen Abständen einen Neustart durch. Daten gingen dadurch verloren.
- **Ursache**: Der binäre Semaphore war nicht mit dem Merkmal zur Unterstützung von Prioritätsvererbung (siehe später) erzeugt worden. Dadurch kam es zur Prioritätsinversion. Ein Watchdog (Timer) erkannte eine unzulässige Verzögerung des Bus Management Prozesses und führte aufgrund eines gravierenden Fehlers einen Neustart durch.

# Ansätze zur Lösung der Prioritätsinversion

- Es existieren verschiedene Ansätze um das Problem der unbegrenzten Prioritätsinversion zu begrenzen:
  - Prioritätsvererbung (priority inheritance)
  - Prioritätsobergrenzen (priority ceiling)
  - Unmittelbare Prioritätsobergrenzen (immediate priority ceiling)
- Anforderungen an Lösungen:
  - leicht zu implementieren
  - Anwendungsunabhängige Implementierung
  - Eventuell Ausschluss von Verklemmungen

# Prioritätsvererbung (priority inheritance)

- Sobald ein Prozess höherer Priorität ein Betriebsmittel anfordert, das ein Prozess mit niedrigerer Priorität besitzt, erbt der Prozess mit niedrigerer Priorität die höhere Priorität. Nachdem das Betriebsmittel freigegeben wurde, fällt die Priorität wieder auf die ursprüngliche Priorität zurück.

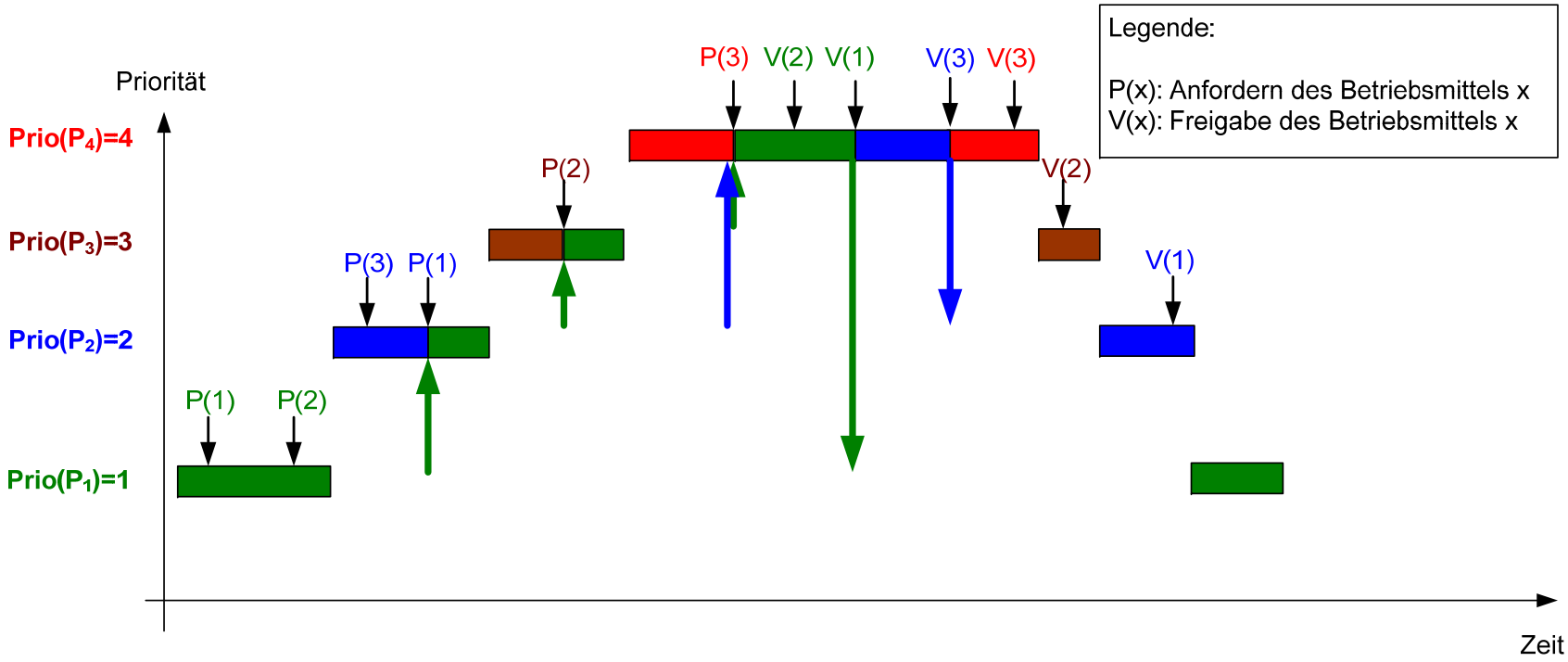
  $\Rightarrow$ Unbegrenzte Prioritätsinversion wird verhindert.

  $\Rightarrow$ Die Dauer der Blockade wird durch die Dauer des kritischen Abschnittes beschränkt.

  $\Rightarrow$ Blockierungen werden hintereinander gereiht (Blockierungsketten).

  $\Rightarrow$ Verklemmungen durch Programmierfehler werden nicht verhindert.

# Beispiel: Prioritätsvererbung



# Prioritätsobergrenzen (priority ceiling)

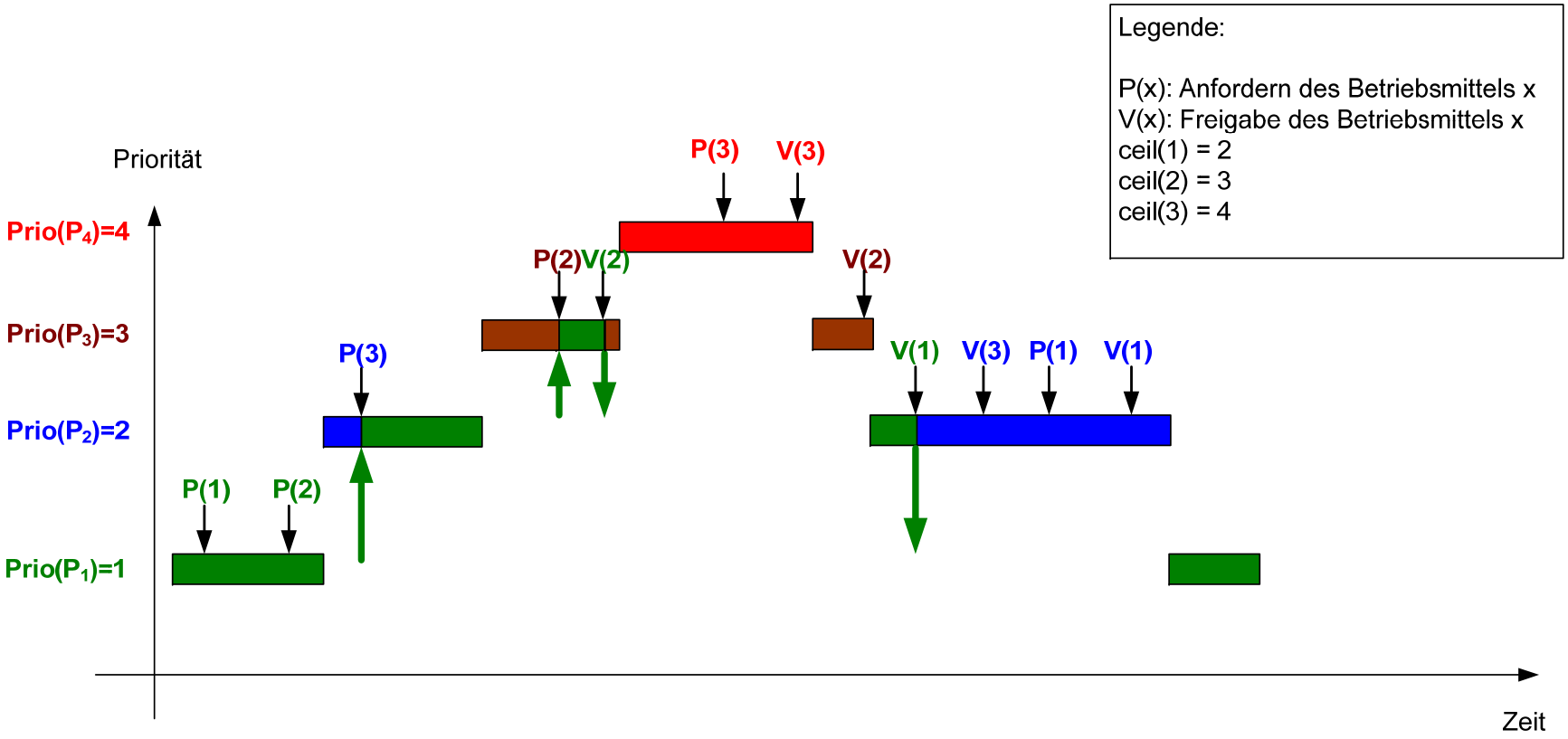
- Jedem Betriebsmittel (z.B. Semaphor) s wird eine Prioritätsgrenze ceil(s) zugewiesen, diese entspricht der maximalen Priorität der Prozesse, die auf s zugreifen.
  - Ein Prozess p darf ein BM nur blockieren, wenn er von keinem anderen Prozess, der andere BM besitzt, verzögert werden kann.
  - Die aktuelle Prioritätsgrenze für Prozess p ist
    aktceil(p) = max{ ceil(s) | s $\in$ locked} mit locked = Menge aller von anderen Prozessen blockierten BM
  - Prozess p darf Betriebsmittel s benutzen, wenn für seine aktuelle Priorität aktprio gilt:
    aktprio(p) > aktceil(p)
  - Andernfalls gibt es genau einen Prozess, der s besitzt. Die Priorität dieses Prozesses wird auf aktprio(p) gesetzt.

  $\Rightarrow$ Blockierung nur für die Dauer eines kritischen Abschnitts

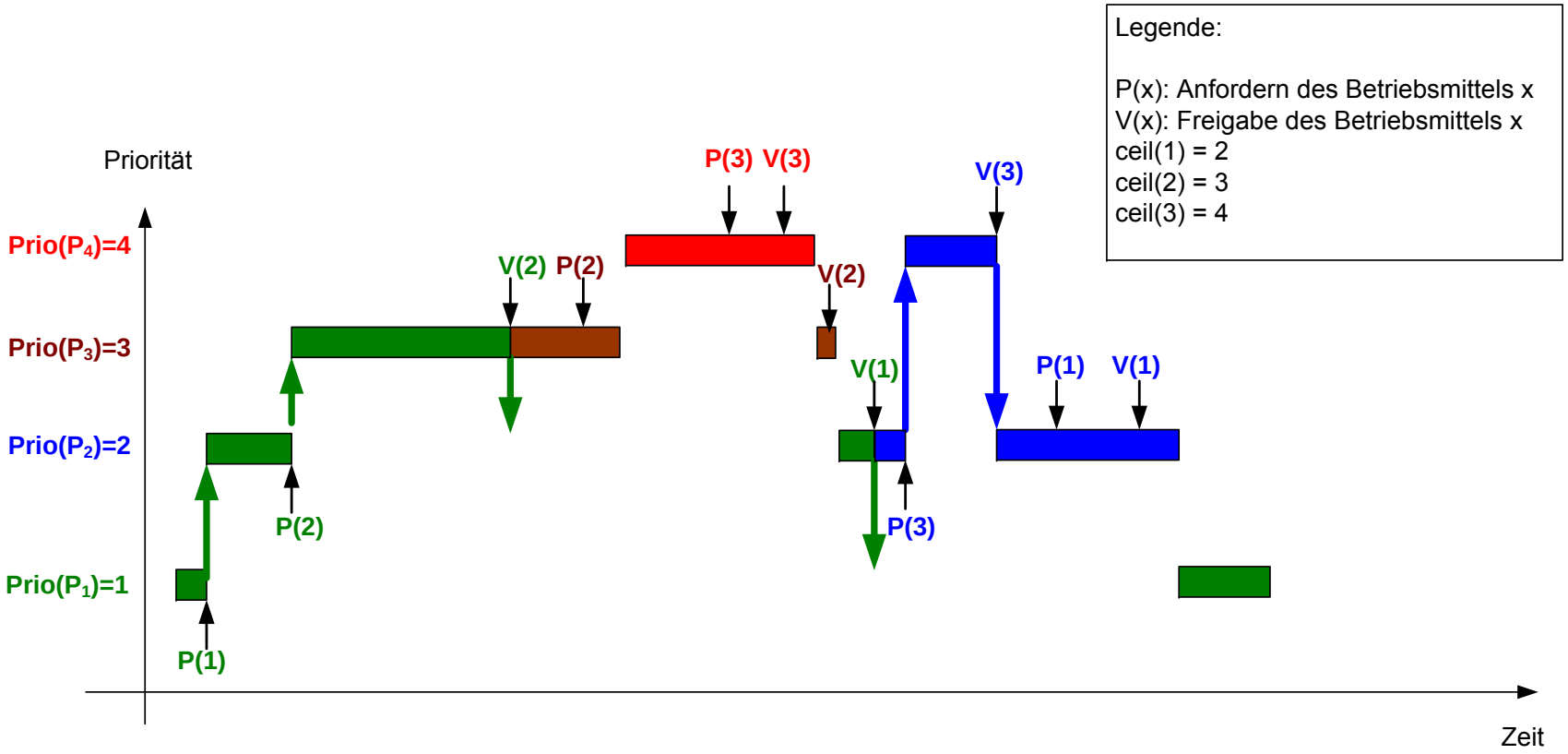
  $\Rightarrow$ Verhindert Verklemmungen

  $\Rightarrow$ schwieriger zu realisieren, zusätzlicher Prozesszustand
- Vereinfachtes Protokoll: **Immediate priority ceiling**: Prozesse, die ein Betriebsmittel s belegen, bekommen sofort die Priorität ceil(s) zugewiesen.

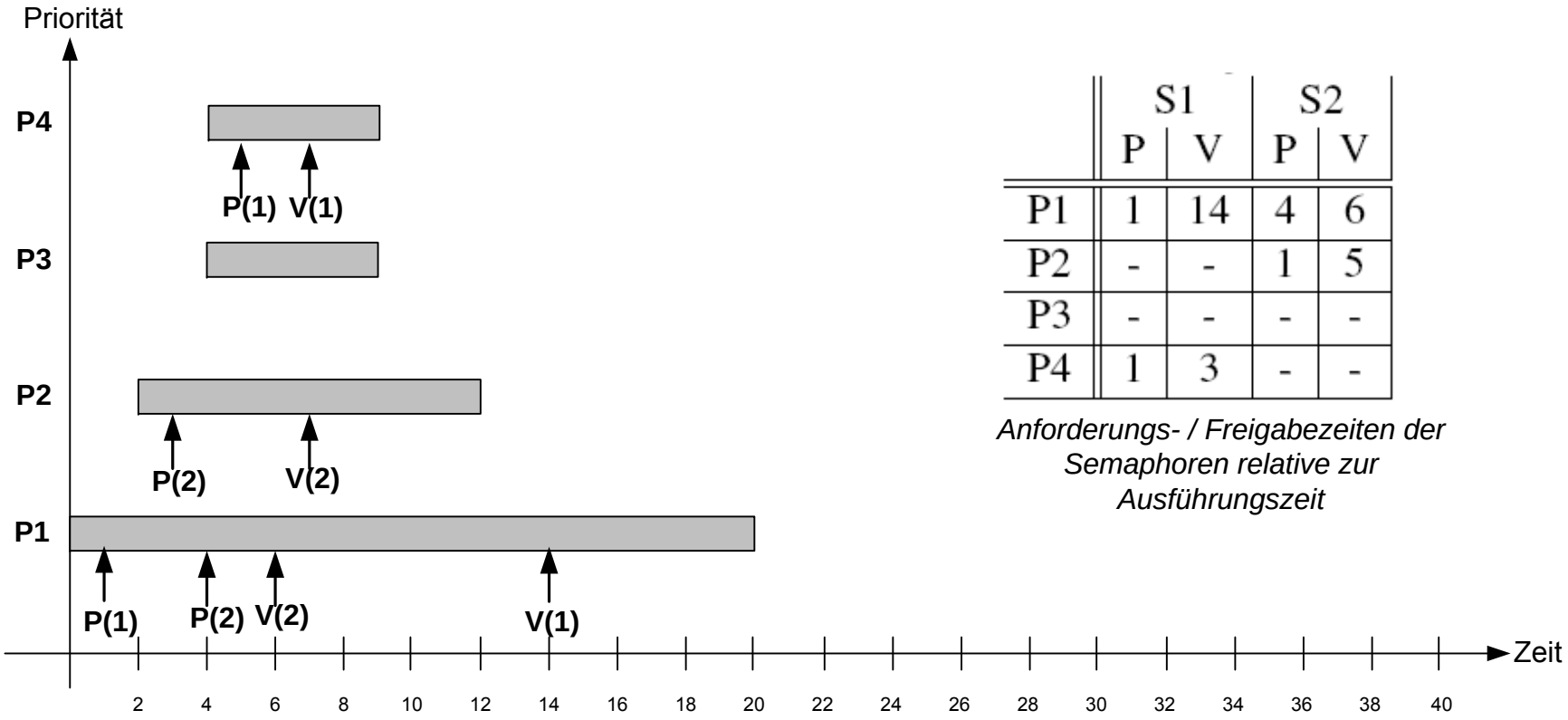
# Beispiel: Prioritätsobergrenzen



# Beispiel: Immediate Priority Ceiling



# Klausur WS 06/07 - Szenario




| | S1 | | S2 | |
|---|---|---|---|---|
| | P | V | P | V |
| P1 | 1 | 14 | 4 | 6 |
| P2 | - | - | 1 | 5 |
| P3 | - | - | - | - |
| P4 | 1 | 3 | - | - |

*Anforderungs- / Freigabezeiten der Semaphoren relative zur Ausführungszeit*